FDP : filtrage et décomposition pour le calcul de plans séquentiels optimaux

Stéphane Grandcolas, Cyril Pain-Barre

FDP : filtrage et décomposition pour le calcul de plans séquentiels optimaux

Stéphane Grandcolas et Cyril Pain-Barre

LSIS – UMR CNRS 6168
Domaine Universitaire de Saint-Jérôme
Avenue Escadrille Normandie-Niemen
13397 MARSEILLE CEDEX 20 France
stephane.grandcolas@lsis.univ-mrs.fr, cyril.pain-barre@lsis.org

Abstract

Dans le domaine de la planification de nombreux systèmes utilisent une représentation de type CSP (*problèmes de satisfaction de contraintes*). Le planificateur FDP que nous présentons dans cet article, fait partie de cette famille. Mais plutôt que de faire appel à un résolveur de CSP indépendant, FDP travaille directement sur une structure similaire au graphe de planification de GRAPHPLAN, appelée *fdp-structure*, avec ses propres mécanismes de filtrage et de décomposition, adaptés au domaine spécifique de la planification. Ainsi pour déterminer s'il existe un plan solution, FDP développe itérativement une fdp-structure jusqu'à trouver une solution ou atteindre une taille limite donnée. A chaque extension de la structure un plan est recherché. Différentes stratégies de recherches ont été envisagées. Actuellement, FDP utilise une recherche avant combinée à une décomposition par partitionnement des ensembles d'actions.

FDP produit des plans séquentiels optimaux. Il intègre diverses techniques pour réduire l'espace de recherche, notamment pour limiter l'énumération de séquences d'actions redondantes, ou pour détecter des situations déja rencontrées. Comparé à d'autres planificateurs séquentiels optimaux sur des jeux de problèmes connus, FDP apparaît compétitif et régulier.

1 Introduction

Les traductions des problèmes de planification dans d'autres domaines de l'Intelligence Artificielle sont nombreuses. La première a été réalisée en logique du premier ordre [6] au tout début des travaux sur la planification.

Suite au remarquable succès du codage de problèmes de planification en problème SAT [10], pour lequel une limite à la taille d'un plan est fixée, de nombreux travaux ont consisté à améliorer ce codage, e.g. [11, 12], ou à mettre au point des traductions dans d'autres formalismes NP-complets, tels que la programmation en nombres entiers [20] ou la satisfaction de contraintes [18, 5, 14]. Parmi ces codages, certains reposent sur le célèbre graphe de planification qui a été introduit avec le système GRAPHPLAN [2]. Celui-ci représente d'importantes relations entre les propositions et les actions du problème de planification et permet d'exhiber des *contraintes d'exclusion mutuelle*, indiquant que certaines actions ne peuvent avoir lieu en même temps, de même que certaines propositions ne peuvent être vraies au même moment. Le graphe ainsi que ces relations ont été codés sous forme de problèmes SAT dans BLACKBOX [12], et sous forme de CSP dans GP-CSP [5]. Ces systèmes utilisent des résolveurs externes afin de résoudre les problèmes générés, et ont donné de très bons résultats.

A contrario, le système DPPLAN [1] implémente une approche SAT sans utiliser de résolveur externe, et exploite le graphe de planification sans en coder les relations. DPPLAN manipule des variables propositionnelles qui représentent les actions et les propositions, et utilise des règles internes pour propager le changement dû à l'affectation d'une valeur à une variable, provoquant éventuellement l'affectation d'autres variables. Le choix d'une variable à affecter ainsi que sa valeur est opéré par une procédure de type Davis et Putnam combinée à plusieurs stratégies possibles. Auparavant, une méthode présentant des similitudes a été introduite dans [16]. Elle utilise des règles de propagation sans toutefois se baser sur un graphe de planification, mène une recherche non-directionnelle et dérive des *invariants* qui sont des clauses binaires devant être satisfaites dans certains états, et qui accroissent les possibilités de propa-

gation des affectations.

Plus récemment, un codage direct des problèmes de planification sous forme de CSP a été proposé [14] pour construire le système CSP-PLAN. Il ne repose pas sur le graphe de planification mais en capture les relations. Cette approche permet de déduire encore d'autres contraintes, telles que les *contraintes binaires additionnelles* et les *contraintes de séquences*. De plus, la réduction de variables mono-valuées ainsi que le raisonnement sur l'ensemble des contraintes permet de réduire le nombre de variables et de contraintes. Cependant, CSP-PLAN fait appel à un résolveur externe pour rechercher un plan solution.

Dans cet article, nous présenton FDP (pour *Filtrage et Décomposition pour la Planification*), un système qui manipule une représentation des problèmes de type CSP mais qui n'utilise aucun résolveur externe. À l'instar de DPPLAN, FDP intègre ses propres règles de consistance ainsi que des mécanismes de filtrage et de décomposition. Ainsi, FDP contrôle totalement la procédure de recherche ainsi que les mécanismes CSP, qui ont été spécifiquement élaborés pour la planification.

FDP travaille sur une structure semblable au graphe de planification de GRAPHPLAN, sans action *no-op*. C'est un graphe par niveaux composé d'étapes. Cette structure est construite incrémentalement jusqu'à trouver une solution ou atteindre une limite fixée du nombre d'étapes. L'implémentation actuelle étend la structure d'une étape supplémentaire lorsqu'aucune solution n'est trouvée pour un nombre d'étapes donné. FDP mène une recherche avant en profondeur d'abord, combinée à la décomposition du problème par partitionnement d'ensembles d'actions. Au final, la recherche d'une solution revient essentiellement en une recherche en profondeur d'abord avec itération de profondeur [13] (ou IDA^* avec une heuristique admissible de coût constant 1). Pour les implémentations futures, d'autres procédures de recherche sont envisagées. En effet, il est possible d'utiliser d'autres heuristiques, admissibles ou non, e.g. [3, 9]. D'autre part, les règles de consistance ainsi que la procédure de filtrage sont déjà adaptées à tout type de décomposition n'importe où dans la structure. Outre la recherche avant, il est tout à fait possible de mener une recherche en arrière ou bidirectionnelle, et plus généralement, une recherche non-dirigée, comme le font les approches SAT ou CSP [16, 1, 14].

Comme d'autres approches similaires [16, 1, 14], FDP ne détecte pas encore l'insolvabilité des problèmes. C'est pourquoi une limite sur la longueur des plans est nécessaire pour s'arrêter lors du traitement d'instance insolvable. Cette lacune de l'algorithme devrait être comblée prochainement.

En revanche, la procédure de recherche est complète et FDP produit des plans séquentiels optimaux en nombre d'actions. Cette caractéristique est devenue assez rare chez les planificateurs récents. Certains mènent des recherches heuristiques sans garantie d'optimalité des solutions [3, 9], d'autres recherchent des plans parallèles optimaux en longueur mais pas en nombre d'actions, e.g. [2, 5, 19].

Nous pensons que l'optimalité en nombre d'actions peut être souhaitable pour certaines applications, en particulier si les actions sont coûteuses. Un futur développement de ce travail sera de tenir compte de coûts des actions et de produire des plans de coût minimal.

Néanmoins, on peut généralement constater que les planificateurs non optimaux en nombre d'actions sont moins sensibles à l'augmentation de la taille des problèmes que les planificateurs optimaux. Bien que nous ayons toutefois observé que sur certains problèmes FDP demeurait compétitif, ces deux types de planificateurs ne peuvent pas être comparés car ils n'ont pas les mêmes objectifs. Une évaluation empirique avec d'autres planificateurs séquentiels optimaux est présentée à la fin de cet article, montrant que FDP est non seulement compétitif mais aussi régulier.

Dans un premier temps, nous introduirons la structure manipulée par FDP puis les règles de consistance. Nous poursuivrons par la description de la procédure de recherche et des méthodes d'élagage de l'arbre de recherche. L'article se termine par les résultats expérimentaux et une conclusion indiquant les perspectives de ce travail.

2 Cadre de travail

L'objectif de la planification est de trouver un *plan*, sous la forme d'un ensemble d'actions prises dans un ensemble donné A, ordonnées totalement ou partiellement, tel que l'exécution à partir d'un *état initial* I de n'importe quelle séquence des actions du plan, respectant l'ordre, satisfait des *buts* G.

De nombreux planificateurs dits parallèles, e.g. [2, 5, 1], renvoient des plans solutions de longueur k de la forme A_0, A_1, ..., A_{k-1}, où chaque A_i est un sous-ensemble de A. De tels plans signifient que chaque action de A_i doit être exécutée avant[1] celles de A_{i+1}. Les actions d'un même ensemble A_i peuvent être exécutée dans n'importe quel ordre ou en parallèle. Dans ce cas, k est appelé le *makespan*, i.e. l'instant le plus tôt où tous les buts seront satisfaits. Dans l'approche que nous proposons ici, chaque A_i est un singleton, i.e. FDP produit des plans séquentiels (séquences d'actions totalement ordonnées). D'autre part, FDP produit des plans optimaux en nombre d'actions, ce qui n'est généralement pas le cas des planificateurs parallèles.

Définition 1 (Problème de planification) *Un problème de planification $\mathcal{P}$ est un quadruplet (A, I, G, P) où A est un ensemble d'actions, I est l'état initial, G représente les buts, et P est l'ensemble de toutes les propositions*

[1] du moins avant celles dont elle introduit une précondition.

qui apparaissent dans I, G et A (sous forme positive ou négative).

L'état initial I et les buts G sont des conjonctions de littéraux. L'ensemble des littéraux est noté L. I indique ce qui doit être vrai dans l'état initial. Il est interprété selon l'*hypothèse du monde clos* : tout proposition qui n'appartient pas à I est considérée fausse dans I.

Le langage d'entrée de FDP est PDDL avec le typage et l'égalité. Étant donné l'état initial d'un problème et l'ensemble des opérateurs du domaine, FDP produit au préalable toutes les propositions et toutes les instances des opérateurs (soit les actions) par l'application de chaque opérateur. Ainsi, le système ne travaille que sur des littéraux et des actions complètement instanciées.

Chaque action $a \in A$ est décrite par ses préconditions $\mathsf{pre}(a)$, et ses effets $\mathsf{eff}(a)$. $\mathsf{pre}(a)$ est un ensemble de littéraux qui doivent être vrais pour que a soit applicable, et $\mathsf{eff}(a)$ est l'ensemble des littéraux que a rend vrais.

FDP utilise une représentation particulière de type CSP pour rechercher des plans valides d'une longueur donnée, appelée *fdp-structure*.

Définition 2 (fdp-structure) *Étant donné un problème de planification $\mathcal{P} = (A, I, G, P)$, une* fdp-structure *pour $\mathcal{P}$ est définie comme un quadruplet $\langle k, V_a, V_p, d\rangle$, où :*

- *k est la taille de la fdp-structure,*
- *$V_a = \{a_0, \ldots, a_{k-1}\}$ est un ensemble de* variables d'actions*,*
- *$V_p = \{x_{i,p}\}_{0 \leq i \leq k, p \in P}$ est un ensemble de* variables de propositions*,*
- *d est une fonction qui met en correspondance les variables et leurs domaines :*
  - *à chaque variable d'action $a_i \in V_a$, correspond un sous-ensemble de A, noté A_i,*
  - *à chaque variable de proposition $x_{i,p} \in V_p$ correspond un sous-ensemble de $\{\text{VRAI}, \text{FAUX}\}$, noté $D_{i,p}$.*

Une sous-structure *d'une fdp-structure $\langle k, V_a, V_p, d\rangle$ est une fdp-structure $\langle k, V_a, V_p, d'\rangle$ telle que pour tout $x \in V_a \cup V_p$, on a $d'(x) \subseteq d(x)$.*

Une variable de proposition dont le domaine est $\{\text{VRAI}, \text{FAUX}\}$ est indéfinie. Si v est la valeur VRAI (resp. FAUX), son *opposé* (noté $opposé(v)$) est la valeur FAUX (resp. VRAI). Par convenance, nous utiliserons aussi des variables de proposition *virtuelles*. À chaque variable de proposition $x_{i,p}$ correspond une variable de proposition virtuelle $x_{i,\neg p}$ dont le domaine est $D_{i,\neg p} = \{opposé(v) \mid v \in D_{i,p}\}$, i.e. $D_{i,\neg p}$ contient les opposés des valeurs possibles de $x_{i,p}$. Par la suite, nous utiliserons des variables de proposition (virtuelles) $x_{i,l}$ où l est un littéral. Les variables virtuelles sont introduites pour alléger cet article, car il sera plus pratique de parler de la valeur de vérité d'un littéral, que de distinguer les cas selon que l'on traite de la valeur VRAI ou FAUX d'une proposition. Il est important de noter que les variables de proposition virtuelles sont des "vues", puisque la suppression d'une valeur v d'un domaine d'une variable de proposition (virtuelle) $x_{i,l}$ entraîne automatiquement la suppression de la valeur $opposé(v)$ du domaine de $x_{i,\neg l}$. Par la suite, le symbole p sera utilisé pour les propositions, et le symbole l pour les littéraux.

Nos fdp-structures sont très proches des graphes de planification de GRAPHPLAN [2]. Ce sont des graphes nivelés qui alternent les *niveaux de propositions* et les *niveaux d'actions*. Le i^{eme} niveau de propositions, noté P_i, est l'ensemble des variables de propositions $\{x_{i,p}\}_{p \in P}$, et représente la validité des propositions à l'étape i. Le i^{eme} niveau d'actions A_i représente les valeurs possibles pour l'action qui sera appliquée à l'étape i. Notons qu'une fdp-structure ne contient pas d'action *no-op*.

Définition 3 (Plan Valide) *Un* plan valide *pour la fdp-structure $\langle k, V_a, V_p, d\rangle$ associée au problème $\mathcal{P} = (A, I, G, P)$ est une affectation θ des variables de $V_a \cup V_p$ telle que :*

1. $\forall a_i \in V_a,\ \theta(a_i) \in A_i$,
2. $\forall x_{i,p} \in V_p,\ \theta(x_{i,p}) \in D_{i,p}$,
3. $\forall l \in I,\ \theta(x_{0,l}) = \text{VRAI}$, *et* $\forall l \notin I,\ \theta(x_{0,l}) = \text{FAUX}$,
4. $\forall l \in G,\ \theta(x_{k,l}) = \text{VRAI}$,
5. $\forall a_i \in V_a, \forall l \in \mathsf{pre}(\theta(a_i)),\ \theta(x_{i,l}) = \text{VRAI}$,
6. $\forall a_i \in V_a, \forall l \in \mathsf{eff}(\theta(a_i)),\ \theta(x_{i+1,l}) = \text{VRAI}$,
7. $\forall l \in L$, *si* $\theta(x_{i,l}) \neq \theta(x_{i+1,l})$ *alors* $l \in \mathsf{eff}(\theta(a_i))$ *ou* $\neg l \in \mathsf{eff}(\theta(a_i))$.

où $\forall p \in P,\ \theta(x_{i,\neg p}) = opposé(\theta(x_{i,p}))$.

La séquence des actions du plan valide (avec exactement une action par étape) est un plan de longueur k pour le problème $\mathcal{P}$. La recherche d'un plan optimal en nombre d'actions consiste à rechercher le plan valide le plus court.

La représentation du problème avec des contraintes sur les variables de V_a et V_p se déduit facilement de la définition d'un plan valide. Par exemple pour exprimer qu'une proposition p qui change de valeur à l'étape i est forcément un effet de l'action appliquée à cette étape (alinéa 7) il faudrait contraindre les variables correspondant à la proposition p aux étapes i et $i+1$ et la variable d'action de l'étape i. Cependant, de telles contraintes sont communes à l'ensemble des problèmes de planification. De telles approches ont déja été faites avec des CSP dynamiques ou des contraintes n-aires [18, 5]. Plutôt que de les générer et les intégrer aux données du problème puis d'utiliser un résolveur CSP, nous avons choisi d'en faire des règles au coeur du système FDP, qui permettent de maintenir la consistance des fdp-structures tout au long de la recherche d'une solution, par élimination de valeurs inconsistantes. FDP partage ce point de vue avec DPPLAN qui utilise des

règles similaires plutôt que des clauses dans le cadre de la planification SAT [1].

3 Fdp-structures consistantes

Nous avons défini dans FDP des règles de consistance afin d'éliminer des fdp-structures des valeurs et des actions qui ne pourront jamais apparaître dans un plan solution. Par exemple une action dont une des préconditions n'est pas valide ne pourra jamais être appliquée et peut être supprimée sans perdre la complétude.

Définition 4 (Inconsistance et variables de proposition) *Etant donnée la fdp-structure $\langle k, V_a, V_p, d\rangle$, la valeur* **VRAI** *de la variable $x_{i,l}$ (i.e. la variable de proposition $x_{i,p} \in V_p$ ou la variable de proposition virtuelle $x_{i,\neg p}$) est inconsistante dans chacune des situations suivantes :*

1. **(persistance avant)** $0 < i \leq k$, VRAI $\notin D_{i-1,l}$ *et* $\forall a \in A_{i-1},\ l \notin \mathsf{eff}(a)$,
2. **(suppression par toutes les actions)** $0 < i \leq k,\ \forall a \in A_{i-1},\ \neg l \in \mathsf{eff}(a)$,
3. **(persistance arrière)** $0 \leq i < k$, VRAI $\notin D_{i+1,l}$ *et* $\forall a \in A_i,\ \neg l \notin \mathsf{eff}(a)$,
4. **(opposé requis par toutes les actions)** $0 \leq i < k, \forall a \in A_i,\ \neg l \in \mathsf{pre}(a)$.

Définition 5 (Inconsistance et variables d'actions) *L'action $a \in A_i$ est inconsistante dans chacune des situations suivantes :*

1. **(précondition non satisfaite)** $\exists l \in \mathsf{pre}(a)$ *tel que* VRAI $\notin D_{i,l}$,
2. **(effets falsifiés)** $\exists l \in \mathsf{eff}(a)$ *tel que* VRAI $\notin D_{i+1,l}$,
3. **(changements non effectifs)** $\exists l \in L$: VRAI $\notin D_{i,l}$, FAUX $\notin D_{i+1,l}$ *et* $l \notin \mathsf{eff}(a)$.

Une valeur d'une variable de proposition qui n'est pas inconsistante est consistante. Une action qui n'est pas inconsistante est consistante.

Définition 6 (Fdp-structure consistante) *La fdp-structure $\langle k, V_a, V_p, d\rangle$ est* consistante *si et seulement si*

1. *les valeurs des domaines des variables de proposition et des variables d'actions sont consistantes,*
2. $\forall p \in P,\ \forall i,\ 0 \leq i \leq k$, *$D_{i,p}$ n'est pas vide,*
3. $\forall i,\ 0 \leq i < k$, *A_i n'est pas vide.*

Propriété 1 (Consistance dans un plan valide) *Etant donné un plan valide θ pour la fdp-structure $\langle k, V_a, V_p, d\rangle$, pour chaque variable $a \in V_a$ l'action $\theta(a)$ est consistante, et pour chaque variable $x \in V_p$ la valeur $\theta(x)$ est consistante.*

Preuve 1 *On vérifie facilement en reprenant les définitions 4 et 5 que dans aucun cas une valeur ou une action inconsistante ne peut apparaître dans un plan valide.*

Propriété 2 (Plus grande sous-structure consistante) *Toute fdp-structure S est équivalente à sa plus grande sous-structure consistante S' si elle existe, et cette sous-structure est unique. Si aucune sous-structure consistante n'existe alors il n'existe aucun plan valide pour S.*

Preuve 2 *Remarquons tout d'abord que si une valeur ou une action est inconsistante dans la structure S, alors elle le reste dans toute sous-structure de S. Puisque qu'une valeur ou une action inconsistante ne peut pas apparaître dans un plan valide, on peut donc l'enlever sans que ça ait d'effets sur l'ensemble des plans solutions. En supprimant itérativement les valeurs et actions inconsistantes jusqu'à ce qu'il n'y en ait plus, on obtient une nouvelle structure S' qui, si elle ne contient pas de domaine vide, est consistante et équivalente à S (dans le sens où S et S' ont le même ensemble de plans solutions). S' est unique puisque les valeurs et actions inconsistantes le restent dans toute sous-structure.*

Les règles de consistance de FDP correspondent assez précisément aux déductions que ferait un mécanisme de filtrage par consistance d'arc, si le problème était représenté avec des contraintes. Cependant nous avons préféré développer une représentation spécifique et une procédure de filtrage basé sur des règles, surtout dans un but d'efficacité et de facilité de mise en place de nos stratégies de recherche, nous verrons pourquoi plus loin.

Exclusions mutuelles

De nombreux systèmes de planification gérent des exclusions mutuelles entre propositions : à une étape donnée deux propositions sont mutuellement exclusives si aucun plan valide ne contient ces deux propositions. Les exclusions mutuelles peuvent être propagées à travers la structure de planification (en général un graphe d'actions de type GRAPHPLAN), en étendant la notion d'exclusion mutuelle aux actions. Pour ces planificateurs, les exclusions mutuelles permettent de construire des ensembles d'actions cohérents, par exemple lors du calcul de plans parallèles, ou encore d'éliminer des propositions qui sont mutuellement exclusives avec des propositions déja instanciées.

Nous n'avons pas implémenté dans FDP de traitement particulier pour les exclusions mutuelles. La raison en est que les exclusions entre actions sont sans objet puisque FDP calcule des plans séquentiels, et les exclusions entre valeurs de variables de propositions sont redondantes avec les règles d'inconsistance. Prenons par exemple les exclusions

du type suivant : les littéraux l et l' sont mutuellement exclusifs à l'étape i, si une des situations suivantes est vérifiée :

1. toute action de A_{i-1} supprime l ou l' ou les deux,
2. l est faux à l'étape $i-1$ et chaque action de A_{i-1} qui ajoute l, supprime aussi l' (ou l'inverse),
3. l et l' sont faux à l'étape $i-1$ et aucune action n'ajoute à la fois l et l'.

Dans chacun des cas on peut démontrer que ces exclusions mutuelles sont sans intérêt. Supposons par exemple dans le cas 2 que l' soit vrai à l'étape i. Alors l ne devrait pas être vrai à l'étape i puisqu'il est mutuellement exclusif avec l'. Si la fdp-structure est consistante, puisque l' est vrai à l'étape i, alors FAUX $\notin D_{i,l'}$ (et VRAI $\notin D_{i,\neg l'}$). Selon la définition 5.2 il n'y a donc pas d'action (consistante) dans A_{i-1} qui supprime l' (i.e. qui ait $\neg l'$ pour effet). Comme toute action de A_{i-1} qui ajoute l supprime aussi l', il n'y a donc pas d'action dans A_{i-1} qui ajoute l. Finalement, puisque VRAI $\notin D_{i-1,l}$, donc VRAI $\notin D_{i,l}$ (définition 4.1).

4 Filtrage des inconsistances

Rendre une fdp-structure donnée consistante consiste à éliminer les valeurs et les actions inconsistantes jusqu'à ce qu'il n'y en ait plus ou qu'un domaine devienne vide.

La similitude avec la consistance d'arcs dans le domaine des CSP, et les techniques de filtrage utilisées pour rendre un problème arc-consistant [4, 15] est évidente. Sur le principe le mécanisme est le même. Dans le cadre des fdp-structures de FDP les contraintes sont très disparates : certaines très simples ou impliquant peu de variables (comme la présence nécessaire des préconditions des actions), d'autres beaucoup plus complexes ou faisant intervenir de nombreuses variables (par exemple pour signifier l'incohérence d'une valeur dont l'opposé serait requis par toutes les actions). Suivant leur type, les différentes contraintes ont un traitement spécifique dans FDP. Pour certaines contraintes des structures de données spécifiques sont maintenues, afin de détecter facilement et rapidement les valeurs inconsistantes. Par exemple, à chaque étape, les faits valides sont référencés dans des listes regroupant entres eux les faits qui sont supprimés par un même nombre d'actions à cette étape. Ces listes sont stockées dans un tableau, indexé sur les nombres d'actions qui suppriment les faits dans chaque liste. Il est très facile à partir de ces listes de déterminer les valeurs qui sont supprimées par toutes les actions à cette étape (il faut le faire notamment chaque fois qu'une action est supprimée). La prise en charge particulière de certaines contraintes dans le processus de filtrage aurait été peu commode si nous avions utilisé un solveur externe.

La fonction **Filtrage** propage un ensemble de suppressions donné dans une fdp-structure S, afin de rendre celle-ci consistante si c'est possible. Les valeurs à supprimer sont insérées dans une file d'attente H. Cette file contient des triplets (p, v, i) représentant des valeurs (la valeur v de l'ensemble $D_{i,p}$) ou des couples (a, i) représentant des actions (l'action a à l'étape i). Tant que la file H n'est pas vide, un élément est enlevé de H, la valeur ou l'action correspondante est supprimée, et les valeurs et les actions qui deviennent alors inconsistantes sont à leur tour insérées dans H. La détection de ces valeurs et actions est directement déduite des règles d'inconsistance de FDP (**Définition 4** et **Définition 5**). De cette façon les suppressions sont propagées vers l'avant et vers l'arrière dans la structure.

```
fonction Filtrage(S, H)
in :  H un ensemble de valeurs et d'actions à supprimer,
      S = ⟨k, V_a, V_p, d⟩ une fdp-structure,
out : VRAI si S est consistant après la propagation
      des suppressions des valeurs et des actions de H,
      FAUX sinon,
1  tant que H ≠ ∅ faire
2      si H contient un triplet (p, v, i) alors
3          enlever (p, v, i) de H,
4          D_{i,p} = D_{i,p} \ {v},
5          si D_{i,p} = ∅ alors
6              renvoyer FAUX,
7      sinon si H contient un couple (a, i) alors
8          enlever (a, i) de H,
9          A_i = A_i \ {a},
10         si A_i = ∅ alors
11             renvoyer FAUX,
12     en se basant sur les règles d'inconsistance de FDP
13     découvrir les valeurs et actions devenues inconsis-
14     tantes et les insérer dans H,
14 renvoyer VRAI.
```

La propagation se termine sur un échec si un domaine devient vide. La fonction **Filtrage** renvoie alors la valeur FAUX. Sinon, lorsque la fonction se termine la fdp-structure S est consistante et la valeur VRAI est renvoyée.

À chaque suppression d'une valeur de proposition, la fonction **Filtrage** recherche des actions inconsistantes, et à chaque suppression d'une action, elle recherche des valeurs de propositions inconsistantes. Dans le pire des cas à chaque étape de la structure, une valeur pour chaque variable de proposition est supprimée, et toutes les actions sauf une sont supprimées. La complexité dans le pire des cas de la fonction **Filtrage** est donc $O(k \times |P| \times |A|)$.

5 Procédure de recherche

Pour trouver un plan optimal, FDP commence par construire une fdp-structure réduite à une seule étape. Cette structure est étendue d'une étape tant qu'aucun plan solution n'est découvert et qu'une borne fixée *a priori* n'est pas atteinte. Chaque fois que la fdp-structure est étendue

fonction **FDP-search**(i, S)
in : $S = \langle k, V_a, V_p, d\rangle$ une fdp-structure consistante, i une étape,
out : VRAI si la fdp-structure contient un plan solution, FAUX, sinon.

```
1  si P_{i+1} ne contient pas de variables indéfinies alors
2      si ⟨P_{i+1}, k − i⟩ ∈ NoGoods alors renvoyer FAUX,
3      si BNA(P_{i+1}, k − (i + 1)) alors renvoyer FAUX,
4  tant que |A_i| = 1 et i < k faire
5      i = i + 1,
6  si i = k alors renvoyer VRAI, (S est un plan solution)
7  ⟨A', A''⟩ = Partitionner(A_i),
8  S' = S,
9  si Filtrer(S', A'') alors
10     si FDP-search(i, S') alors
11         renvoyer VRAI,
12 S' = S,
13 si Filtrer(S', A') alors
14     si FDP-search(i, S') alors
15         renvoyer VRAI,
16 si P_{i+1} ne contient pas de variables indéfinies alors
17     NoGoods = NoGoods ∪{⟨P_{i+1}, k − i⟩},
18 renvoyer FAUX.
```

une procédure de recherche en profondeur détermine s'il existe un plan solution. Ceci assure l'optimalité de la solution si elle existe. L'approche utilisée pour la recherche d'un plan solution est de type *diviser pour régner* : la fdp-structure est décomposée en sous-structure plus petites qui sont traitées récursivement. Chaque fois les sous-structures sont filtrées afin de détecter les échecs le plus tôt possible.

Nous avons essayé plusieurs décompositions telles que l'*énumération des actions*, l'*affectation des propositions*, ou le *partitionnement des ensembles d'actions*. Cette dernière stratégie donne les meilleurs résultats : le principe est de partitionner l'ensemble des actions d'une étape donnée, de façon à mettre ensemble les actions qui ont des suppressions communes. La procédure de partitionnement recherche une proposition telle que le nombre d'actions qui la suppriment soit aussi proche que possible du nombre d'actions qui ne la suppriment pas. De cette façon on génère deux ensembles d'actions de tailles aussi proches que possibles. On espère ainsi générer des sous-problèmes équilibrés, ou en tout cas ne pas produire un sous-problème qui soit aussi difficile que le problème à décomposer.

Lors de la recherche d'un plan de longueur k, FDP utilise une fdp-structure S et un ensemble *NoGoods*. Initialement chaque ensemble d'actions contient toutes les actions de A, les variables de proposition sont toutes indéfinies, et les valeurs qui ne sont pas dans l'état initial ainsi que les valeurs opposées aux buts sont supprimées. Ensuite un filtrage préliminaire est appliqué à la structure S. Si S est inconsistant la recherche s'arrête sur un échec, il n'existe pas de plan solution de longueur k. Sinon la procédure **FDP-search** est appliquée à la fdp-structure consistante $S : S$ est décomposée en deux sous-structures correspondant aux sous-ensembles d'actions obtenus par partition d'un ensemble d'actions (ligne 7). Chaque fois, l'ensemble à partitionner est le premier ensemble d'actions en partant de l'état initial, qui ne soit pas réduit à un singleton (lignes 4 et 5). **FDP-search** est donc une recherche en profondeur d'abord en partant du début. Le plan en construction est étendu étape par étape jusqu'à ce que la fdp-structure devienne inconsistante ou qu'un plan valide soit découvert.

Pendant la recherche, chaque fois que **FDP-search** rencontre un échec à partir d'un état qui ne contient pas de variables indéfinies, cet état est mémorisé dans l'ensemble *NoGoods* (lignes 16 et 17), avec la distance qui sépare cet état de l'état final ($k-i$ est la distance de l'état P_{i+1} à l'état final). En effet, l'état P_{i+1} pourrait peut-être être étendu à un plan solution s'il y avait plus d'étapes. Chaque fois que P_{i+1} sera rencontré pendant la recherche à une distance de l'état final inférieure à $k - i$, la recherche sera abandonnée. L'ensemble *NoGoods* est initialisé à l'ensemble vide une fois pour toutes avant la construction de la première fdp-structure. La mémorisation des états invalides améliore considérablement les performances de la recherche.

Les heuristiques habituellement utilisée dans le domaine des CSP pour le choix des variables conduiraient sans doute à choisir des variables de proposition, car ces variables ont des domaines réduits, et probablement les variables indéfinies les plus proches de l'état initial, car ce sont les plus contraintes (l'état initial étant complètement instancié, le début de la structure est plus contrainte que la fin, où l'état final est en général partiellement indéfini, et cet état de fait perdure pendant la recherche). La technique de décomposition par partitionnement utilisée dans FDP pourrait tout à fait être appliquée aux CSP (à l'origine elle leur était destinée). Son efficacité provient du fait qu'en partitionnant un ensemble d'actions on n'a pas à faire le choix d'une action particulière, qui pourrait s'avérer un mauvais choix, tout en profitant de la propagation des suppressions communes, puisque les actions sont regroupées en fonction de leurs suppressions (ou des suppressions qu'elle ne font pas).

Nombre minimal d'étapes

Chaque fois qu'un niveau P_i est complètement instancié, FDP effectue une évaluation gloutonne du nombre minimal d'étapes nécessaires pour atteindre les buts. L'idée est de choisir pour chaque étape suivant P_i une action qui ajoute le plus de buts manquants à l'étape i. Si le nombre de buts non satisfaits à l'étape i et ajoutés par les actions sélectionnées est moindre que le nombre de buts manquants dans P_i, c'est qu'il n'est pas possible d'obtenir tous les buts à partir de l'état P_i (ligne 3, la fonction **BNA** (pour ButsNonAtteignables) détermine si les buts manquants dans P_{i+1} sont

impossibles à atteindre en $k - (i + 1)$ étapes).

Séquences redondantes d'actions

Du fait que FDP produit des plan séquentiels, il peut générer de multiples permutations d'une même séquence d'actions, effectuant alors des traitements redondants et sans intérêt. Ceci constitue le plus gros défaut des planificateurs séquentiels comparé aux planificateurs parallèles. Afin de remédier en partie à ce problème, FDP ne considère pas les séquences d'actions *indépendantes* qui ne respectent pas un ordre total arbitraire sur les actions noté $\prec$.

Définition 7 (2-Séquences ordonnées) *Les actions a_1 et a_2 sont* indépendantes *si elles satisfont les propriétés suivantes :*

1. $\forall l \in \mathsf{pre}(a_1)$, $l \notin \mathsf{eff}(a_2)$ *et* $\neg l \notin \mathsf{eff}(a_2)$[2],
2. $\forall l \in \mathsf{pre}(a_2)$, $l \notin \mathsf{eff}(a_1)$ *et* $\neg l \notin \mathsf{eff}(a_1)$.

La séquence d'actions (a_1, a_2) est une 2-séquence ordonnée *si a_1 et a_2 sont indépendantes et $a_1 \prec a_2$, ou si a_1 et a_2 ne sont pas indépendantes.*

Dans FDP les 2-séquences non ordonnées ne sont pas prises en compte, de même que les séquences constituées d'actions qui ont des effets exactement opposés, i.e. telles que $\mathsf{eff}(a_1) = \{\neg l \mid l \in \mathsf{eff}(a_2)\}$, puisque ces séquences sont sans objet dans un plan optimal.

Propriété 3 *Etant donné un plan $(a_0, \ldots, a_{k-1})$, il existe une permutation des actions $a_0, \ldots, a_{k-1}$ qui est un plan et telle que tout couple d'actions successives constitue une 2-séquence ordonnée.*

Preuve 3 *Il suffit de permuter autant que possible les actions successives indépendantes qui ne sont pas ordonnées. Chaque transformation produit un plan qui est équivalent au plan initial. Quand le processus se termine chaque couple d'actions successives est une 2-séquence ordonnée.*

Pour implémenter cette propriété, les règles suivantes ont été ajoutées à la définition 5 des actions inconsistantes :

4. **(non-ordonnée à gauche)**
   $i > 0$ et $\forall a' \in A_{i-1}$, (a', a) n'est pas une 2-séquence ordonnée,
5. **(non-ordonnée à droite)**
   $i < k - 1$ et $\forall a' \in A_{i+1}$, (a, a') n'est pas une 2-séquence ordonnée.

[2]Si a_1 requiert un fait qui est ajouté par a_2, il est possible dans certains cas que la séquence (a_2, a_1) doive être autorisée. Donc a_1 et a_2 ne peuvent pas être considérées comme indépendantes.

Actions et littéraux pertinents

Les actions qui ne peuvent pas aider effectivement à atteindre les buts ne sont d'aucune utilité pour le calcul de plans optimaux. Ceci est notamment le cas des actions de la dernière étape qui n'ajoutent aucun but, et ces actions peuvent être supprimées sans perdre la complétude. Cette propriété peut être propagée vers l'état initial à travers la notion de *littéraux et actions pertinents*.

Définition 8 (Actions et littéraux pertinents) *Les* actions pertinentes *et les* littéraux pertinents *sont définis récursivement dans une fdp-structure donnée, de la façon suivante :*

1. *tout but de G est pertinent,*
2. *un littéral l est pertinent à l'étape i si il existe une action pertinente $a \in A_i$ qui ait l comme précondition,*
3. *une action a est pertinente à l'étape i si un de ses effets est pertinent à l'étape $i + 1$.*

Les actions qui ne sont pas pertinentes à une étape donnée peuvent être supprimées puisqu'elles ne participeront jamais à un plan solution optimal. A tout moment, la fonction **Filtrer** s'occupe d'éliminer les actions non pertinentes.

6 Résultats expérimentaux

Nous avons comparé FDP avec d'autres planificateurs sur de nombreux problèmes usuels en planification. Il s'agit de problèmes divers tels que *mystery*, *mystery-prime*, *hanoi towers* et *xy-world*, de probèmes issus de la compétition IPC-3 (*depot*, *driver-log*, *freecell*, *satellite* et *zeno-travel*), et d'autres issus de la compétition IPC-4 (*airport*, *psr-small*, *pipesworld* (*tank-notemp*, *notank-notemp*), *optical*, *philosophers* et *satellite*). Dans chaque série nous avons enlevé les problèmes trop faciles (i.e. que tous les planificateurs traitent en moins de 0,1 seconde) ainsi que les problèmes qu'aucun planificateur séquentiel n'a pu traiter en moins de 1500 secondes (le temps maximal autorisé pour le calcul dans nos expérimentations). Ensuite nous avons conservé uniquement les problèmes représentatifs dans chaque série : par exemple dans les séries *mystery* ou *psr*, de nombreuses instances produisent des résultats très semblables, et nous ne les avons donc pas tous présentés ici.

Comme nous l'avons dit précédemment, le partitionnement d'ensembles d'actions est le mode de décomposition qui donne les meilleurs résultats. C'est cette décomposition qui est utilisée pour les versions **fdpN2s** et **fdp2s** de FDP qui ont été testées. La seule différence entre ces deux versions est l'élimination des 2-séquences non ordonnées pour **fdp2s**. Dans quelques cas le calcul et la détection des 2-séquences non ordonnées est plus coûteux que les bénéfices obtenus. Il y a plusieurs raisons à cela : tout d'abord le

	actions	prop.	long.	mkspan	fdpN2s	fdp2s	bfhspbk	bfhspfw	hspSeq	satplanner	dpplan	satplan
mprime-x-5	3464	319	11	6	1500,00	1500,00	246,77	1500,00	1500,00	136,28	2,53	–
mprime-x-7	1708	426	5	1	0,54	1,10	282,63	13,49	15,73	8,11	1,15	0,18
mprime-x-8	10206	513	6	5	18,82	64,71	1500,00	1500,00	139,2	62,15	–	1,80
mprime-x-21	15324	838	6	5	13,84	123,96	1500,00	1500,00	244,6	–	6,88	–
mprime-x-29	4872	269	4	4	1,57	7,81	259,66	51,69	63,38	10,74	1,32	1,04
mystery 19	6521	562	6	6	3,03	17,25	1139,77	403,15	54,09	59,32	3,28	17,86
mystery 30	4085	412	9	6	44,39	20,73	392,35	1500,00	44,49	41,44	3,08	26,06
hanoi 6	166	75	63		3,67	2,53	0,46	0,18	0,38	1500,00	1500,00	1500,00
hanoi 7	238	94	127		33,71	23,56	–	–	1,72	1500,00	1500,00	1500,00
xy-world f10	200	40	10	1	0,05	0,06	74,08	270,29	–	8,52	0,01	0,18
Cumul divers					**1619,62**	**1761,71**	**5395,70**	**6738,80**	**2063,59**	**3326,50**	**3018,20**	**3047,10**
depot 2	180	82	15	8	3,56	2,05	0,46	6,35	4,29	3,89	0,08	0,50
Driver log 4	144	63	16	7	55,28	23,42	1,09	113,90	44,97	213,38	0,44	0,45
Driver log 5	168	75	18	8	205,13	103,66	27,89	896,25	1500,00	990,14	0,07	0,65
FreeCell 2	1144	139	14	8	109,56	100,61	6,27	347,27	1500,00	1413,84	31,20	1500,00
FreeCell 3	1616	183	18	7	1500,00	1203,77	94,22	1500,00	1500,00	1500,00	2,01	1500,00
satellite 3	188	66	11	6	1,18	0,98	0,21	138,81	0,26	0,93	0,04	–
satellite 4	259	71	17	10	87,31	62,18	4,33	1500,00	33,15	99,15	17,51	–
Zeno 5	376	52	11	5	5,27	3,45	0,41	41,91	–	60,32	0,07	0,46
Zeno 6	392	58	11	5	40,25	20,4	1,96	297,84	–	53,31	0,14	2,65
Zeno 7	408	64	15	6	35,52	25,62	13,77	772,86	–	1500,00	0,08	1,02
Cumul IPC3					**2042,32**	**1546,14**	**150,60**	**5615,10**	**4582,67**	**5834,90**	**51,60**	**3005,70**
Airport 12	203	356	39	21	4,09	2,44	11,49	0,77	3,25	159,55	0,15	1,13
Airport 14	347	493	60	26	579,78	200,02	225,78	35,26	333	1500,00	65,32	3,30
PSR 19	163	41	25	15	6,64	5,79	42,79	1,62	1500,00	25,77	549,18	1,79
PSR 22	112	56	33	25	75,44	87,15	102,42	135,72	885,79	106,56	1500,00	168,86
PSR 25	9400	58	9	9	27,66	49,64	22,56	126,41	363,42	27,37	–	132,44
PSR 36	343	82	22	16	848,14	519,1	128,20	1500,00	1500,00	11,72	1500,00	7,16
PSR 40	762	66	20	15	259,38	216,73	31,39	396,39	1500,00	11,10	1500,00	18,09
PSR 46	98	60	34		41,68	39,88	996,85	79,30	1500,00	254,20	1500,00	1500,00
PipesW NT-NT 11	720	211	20		195,97	211,31	720,98	278,85	1500,00	1500,00	1500,00	1500,00
PipesW NT-NT 13	896	251	16	12	102,21	88,74	62,22	147,93	1500,00	160,61	287,99	81,64
PipesW NT-NT 21	1140	292	14	14	19,83	7,48	70,91	104,29	1500,00	20,53	1500,00	200,82
PipesW T-NT 5	768	164	8	7	2,97	1,59	–	74,03	198,19	13,90	1,41	3,46
PipesW T-NT 6	768	164	10	7	15,89	10,85	–	301,49	1500,00	66,38	2,06	9,42
PipesW T-NT 7	2672	204	8	7	32,47	17,33	–	1500,00	1500,00	–	1500,00	512,50
Optical 1	446	282	36	13	148,13	121,63	–	34,79	1500,00	1500,00	0,18	1,74
Philosophers 3	112	120	44	11	136,96	145,64	–	12,33	1500,00	1500,00	0,16	0,56
satellite 3	188	66	11	6	1,17	0,97	0,21	165,05	0,25	0,97	0,04	7,57
satellite 4	259	71	17	10	87,39	62,39	3,90	1500,00	33,27	85,75	19,48	165,89
Cumul IPC4					**2585,80**	**1788,68**	**2419,70**	**6394,20**	**18317,17**	**7544,40**	**11425,90**	**4316,40**

FIG. 1 – Temps CPU pour divers planificateurs sur une sélection de problèmes (les temps sont en secondes sur une machine Linux dotée d'un procésseur Pentium 3 Ghz et de 1 Go de RAM).

calcul préliminaire des 2-séquences est polynomial avec le nombre d'actions, et ce calcul peut donc être assez coûteux. C'est le cas notamment pour la série *mystery*, dans laquelle les nombres d'actions sont élevés. Ensuite le gain produit par l'élimination des 2-séquences non ordonnées est d'autant plus faible que celles-ci sont (relativement) peu nombreuses. Enfin l'élimination d'actions qui ne peuvent apparaître dans des 2-séquences ordonnées, peut modifier le comportement de certaines heuristiques par exemple pour le partitionnement des ensembles d'actions, et avoir un impact négatif sur les performances.

Mis à part ces cas isolés, il semble que **fdp2s** soit meilleur que **fdpN2s**, même si la différence n'est pas très importante.

Nous avons inclus les résultats obtenus par les planificateurs BFHSP [21] avec une recherche en arrière et l'option h3max (**bfhspbk**), BFHSP avec une recherche en avant et l'option h1max (**bfhspfw**), HSP [8] avec les options `-seq -bfs` afin de générer des plans optimaux séquentiels (**hspSeq**), SATPLANNER [17] un planificateur basé sur des calculs de satisfiabilité de formules propositionelles muni du solveur SAT Siege V4, avec l'option `-opt` pour le calcul de plans séquentiels optimaux, DPPLAN [1] qui utilise des règles de propagation, très proches de celles présentées dans cet article, dans une procédure de recherche de type Davis et Putnam, et SATPLAN [12] avec le solveur SAT Siege V4. Ces deux derniers produisent des plans parallèles, tandis que les quatre premiers produisent des plans séquentiels optimaux. Nous avons signalé avec le symbole – les erreurs produites par les versions des planificateurs dont nous disposons (trop d'opérateurs, erreur d'analyse, nombre maximal d'étapes atteint, planificateur tué, temps ou capacité mémoire dépassés,...). La limite de temps a été fixée à 1500 secondes. Nous avons aussi indiqué les temps de calcul cumulés pour chaque série de domaines (divers, IPC-3, IPC4)[3].

De façon générale, les planificateurs parallèles donnent de meilleurs résultats que les planificateurs séquentiels, probablement du fait que les plans parallèles sont plus

[3]les erreurs (–) comptent pour un temps de calcul de 0s.

courts. Cela dit, ces planificateurs ne sont pas comparables aux planificateurs séquentiels, car ils ne cherchent pas la même optimalité. Nous avons tout de même publié les résultats de DPPLAN et SATPLAN car ils servent de référence. En particulier, DPPLAN est un planificateur qui utilise des règles de propagation très semblables à celles que nous avons présentées ici.

Au vu des résultats il semble que FDP et BFHSP soient meilleurs que les autres planificateurs séquentiels testés. Généralement, FDP est le meilleur quand le nombre d'actions est important et la longueur des plans solutions petites, comme le domaine *mystery* par exemple. Sur ces problèmes il semble qu'une recherche en largeur d'abord sera pénalisée par le traitement d'ensembles d'états très importants. A l'inverse, quand il y a peu d'actions et que les plans sont longs, BFHSP domine généralement, comme pour les domaines *Hanoi*, *Driver Log* et *FreeCell*. Cependant FDP est tout de même meilleur dans la série *PSR*. Nous nous intéresserons aux raisons de cette supériorité dans la suite de nos recherches.

On peut aussi remarquer que **bfhsp-back** et **bfhsp-forw** ont très rarement de bons résultats en même temps, alors que **fdp2s** par exemple est souvent meilleur qu'au moins une des versions de BFHSP. Finalement, la principale qualité de FDP est sa régularité. Par exemple, si DPPLAN et **bfhsp-back** sont vraiment meilleurs dans la série IPC-3, ils ont de plus mauvais résultats dans d'autres séries, et ont même dépassé parfois le temps limite (certains planificateurs qui n'ont pu être arrété convenablement n'ont pas produit de résultat après plus de 10000 secondes de calcul). Ceci est rarement le cas de FDP qui n'a dépassé le temps limite qu'une seule fois sur l'ensemble des problèmes sélectionnés.

7 Conclusion et perspectives

Nous avons présenté dans cet article FDP, un nouveau planificateur pour le calcul de plan séquentiels optimaux. Comparé aux autres planificateurs séquentiels optimaux FDP est plutôt performant. Ses règles de consistance et sa stratégie de décomposition lui permettent d'effectuer une recherche en avant, en arrière, bidirectionelle ou plus généralement une recherche non dirigée. FDP pourrait être amélioré en utilisant une évaluation plus fine de la distance aux buts [7] ou en effectuant parallèlement une recherche en avant et une recherche en arrière coopérant par l'intermédiaire des ensembles d'états valides ou invalides. Une amélioration majeure sera de mettre en place un critère de terminaison pour les problèmes sans solution. FDP sera aussi étendu afin de prendre en compte les actions valuées et de calculer des plans de coûts minimaux. Enfin la planification avec des ressources limitées constitue un développement prometteur du système FDP.

Références

[1] Marco Baioletti, Stefano Marcugini, and Alfredo Milani. Dpplan : An algorithm for fast solutions extraction from a planning graph. In *AIPS*, pages 13–21, 2000.

[2] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.

[3] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2) :5–33, 2001.

[4] Rina Dechter. *Constraint Processing.* Morgan Kaufmann, San Francisco, 2003.

[5] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction : Solving the planning graph by compiling it into csp. *Artif. Intell.*, 132(2) :151–182, 2001.

[6] C. Cordell Green. Application of theorem proving to problem solving. In *IJCAI*, pages 219–240, 1969.

[7] Patrik Haslum, Blai Bonet, and Hector Geffner. New admissible heuristics for domain-independent planning. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 1163–1168. AAAI Press AAAI Press / The MIT Press, 2005.

[8] Patrik Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *AIPS*, pages 140–149, 2000.

[9] Jörg Hoffmann and Bernhard Nebel. The FF planning system : Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14 :253–302, 2001.

[10] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *ECAI*, pages 359–363, 1992.

[11] Henry A. Kautz and Bart Selman. Pushing the envelope : Planning, propositional logic and stochastic search. In *AAAI/IAAI, Vol. 2*, pages 1194–1201, 1996.

[12] Henry A. Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *IJCAI*, pages 318–325, 1999.

[13] R.E. Korf. Macro-operators : A weak method for learning. *Artificial Intelligence*, 26(1) :35–77, 1985.

[14] Adriana Lopez and Fahiem Bacchus. Generalizing graphplan by formulating planning as a CSP. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 954–960. Morgan Kaufmann, 2003.

[15] A.K. Mackworth. Consistency in networks of relations. In *Artificial Intelligence*, pages 8 :99–118, 1977.

[16] Jussi Rintanen. A planning algorithm not based on directional search. In *KR*, pages 617–625, 1998.

[17] Jussi Rintanen. Evaluation strategies for planning as satisfiability. In Ramon López de Mántaras and Lorenza Saitta, editors, *ECAI*, pages 682–687. IOS Press, 2004.

[18] Peter van Beek and Xinguang Chen. Cplan : A constraint programming approach to planning. In *AAAI/IAAI*, pages 585–590, 1999.

[19] Vincent Vidal and Hector Geffner. Branching and pruning : An optimal temporal pocl planner based on constraint programming. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 570–577. AAAI Press / The MIT Press, 2004.

[20] Thomas Vossen, Michael Ball, Amnon Lotem, and Dana S. Nau. On the use of integer programming models in ai planning. In *IJCAI*, pages 304–309, 1999.

[21] Rong Zhou and Eric A. Hansen. Breadth-first heuristic search. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *ICAPS*, pages 92–100. AAAI, 2004.